Final

- Please read all instructions (including these) carefully.
- There are eight questions on the exam, some with multiple parts. You have 2 hours to work on the exam.
- The exam is open book, open notes. *If you use a tablet, make sure that wifi is off. During the exam, you may only refer to the course textbooks, lecture notes, sample exams, and other printed materials. Any violations will be considered plagiarism.*
- Please write your answers in the space provided on the exam, and clearly mark your solutions.
- Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. You may get as few as 0 points for a question if your solution is far more complicated than necessary. Partial solutions will be graded for partial credit.

NAME: _____
First Name Last Name SID

Problem	Max. points	Points
1	15	
2	12	
3	20	
4	10	
5	24	
6	10	
7	9	
8 (**EC**)	10	
TOTAL	100 + 10	

1. Inheritance and subtyping in object-oriented languages (15 points)

   (a) (5 points) Virtual method tables

   We have four classes: A, B, C, and D. The following are contents of their virtual method tables (VMTs):

```
A's VMT                    B's VMT
--------                   --------
   D::m1                      B::m1
   A::m2                      D::m2
   A::m3                      B::m3

C's VMT                    D's VMT
--------                   --------
   C::m1                      D::m1
   D::m2                      D::m2
   B::m3
   C::m4
```

   Draw the inheritance graph to relate classes A, B, C, and D (put the base class at the top).

(b) (10 points) Does the following Java code compile? Circle **YES** or **NO**.

i.
```
class A {
  public A foo() { return this; }
}
class B extends A {
  @Override
  public B foo() { return this; }
}
```

YES **NO**

ii.
```
class A {
  public A foo(A a) { return this; }
}
class B extends A {
  @Override
  public B foo(A a) { return this; }
}
```

YES **NO**

iii.
```
class A {
  public A foo(A a) { return this; }
}
class B extends A {
  @Override
  public B foo(B b) { return this; }
}
```


YES **NO**

iv.
```
class A {
  public A foo(B b) { return this; }
}
class B extends A {
  @Override
  public B foo(B b) { return this; }
}
```

YES **NO**

v.
```
class A {
  public A foo(B b) { return this; }
}
class B extends A {
  public B foo(A a) { return this; }
}
```

YES **NO**

2. Semantics of LISP programs (12 points)

Evaluate the following s-expressions and show the output of the last s-expression. *Circle your answers.*

(a) (1 point)

```
(* (- (+ 3 4) 5) 6)
```

(b) (1 point)

```
(car (cdr (cdr '(1 (2 4) 8))))
```

(c) (3 points)

```
(do ((l1 '(5 4 3 2 1 0) (cdr l1))
     (l2 '(0 1 2 3 4 5) (cdr l2))
     (cnt 0 (+ 1 cnt)))
    ((< (car l1) (car l2))
     cnt))
```


(d) (3 points)

```
(setf fn1 (let ((a 1)) #'(lambda () (setf a (+ a 2)))))
(setf fn2 #'(lambda () (let ((a 1)) (setf a (+ a 2)))))
(list (funcall fn1) (funcall fn1) (funcall fn2) (funcall fn2))
```

(e) (4 points)

```
(defun foo (f l)
  (if (null l) ()
    (let ((lnew (mapcar f l)))
      (cons (car lnew) (foo f (cdr lnew))))))
(foo #'(lambda (x) (+ x 2)) '(1 3 5 7))
```

3. LISP function definitions (20 points)

   In this problem, you are asked to write a few LISP functions to perform certain tasks.

   (a) (6 points) Write a function, `mymaplist`, to implement the built-in `maplist` function.

   (b) (7 points) Write a function, `quicksort`, to implement the QuickSort algorithm. It should take a list `l` as argument and return the sorted list of `l` in increasing order of its elements.

```
> (quicksort '(3 7 5 2 6 4))
(1 2 3 4 5 6 7)
```

(c) (7 points) Write a function, `subsequence`, that takes as arguments two lists of numbers or symbols `l1` and `l2` and returns true if `l1` is a **consecutive subsequence** of `l2`.

```
> (subsequence '(a b) '(c a b d))
T
> (subsequence '(a b) '(c a c b d))
NIL
```

4. Prolog search and backtracking (10 points)

The following describes a set of rules and facts:

rule 1	`a(X,Y) :- c(Y), b(X), d(Y,X).`
rule 2	`b(1).`
rule 3	`b(2).`
rule 4	`c(3).`
rule 5	`d(2,1).`
rule 6	`d(3,2).`

On the lines below, give a step by step description of how Prolog evaluates the following query:

```
| ?- a(X,Y).
```

Your answer must be descriptive. At each step show the following: (i) the subgoals at the step; (ii) the rule that will be applied (write "none" if no of the rules applies); (iii) variables instantiations, if any; and (iv) comments that indicate if there was a "match", "success", "failure", or "backtrack". Clearly mark the line where "success" is achieved and show the output generated.

Also, when backtracking, clearly show the goals to which the system backtracks. Continue on the back of the page if you need more space.

step no.	current subgoals	rule applied	variable instantiation	comments
1	`a(X,Y)`	1		match
2	`c(Y), b(X), d(Y,X)`	4	`Y=3`	match

5. Prolog programs (24 points)

   (a) (6 points) What do the following Prolog queries output given their associated facts and rules?

      i. (2 points)

```
| ?- (X = 0 ; X = 1), !, X > 0.
```

      ii. (2 points)

```
foo([H, _ | R], [H | T]) :- !, foo(R, T).
foo(_, []).
| ?- foo([1,2,3,4,5], X).
```


      iii. (2 points)

```
len([],0).
len([_|T],R) :- len(T,R1), R is R1+1.
has(X,[X|_]) :- !.
has(X,[_|T]) :- has(X,T).
| ?- has(0,X), !, len(X,1).
```

(b) (6 points) Write a predicate, `rmdup(L, N)`, that takes two lists `L` and `N` such that `N` is the same as `L` except all duplicates have been removed. An example usage of `rmdup` is shown below:

```
| ?- rmdup([a,b,r,a,c],N).
N = [a,b,r,c] ? ;
no
```


(c) (6 points) Write a predicate, `rpn(L,N)`, that implements a reverse polish calculator over integers supporting the four standard binary operators (*i.e.*, +, −, ∗, and /), where `L` is a list representation of the expression and `N` is the evaluation result.


```
| ?- rpn([2,3,'+'], N).
N = 5 ? ;
no
```



```
| ?- rpn([2,3,'*',4,'-'], N).
N = 2 ? ;
no
```

The two examples correspond to "2 + 3" and "(2 * 3) - 4" in infix notation, respectively.

(d) (6 points) Write predicates to implement the common Boolean operators `and(X,Y)`, `or(X,Y)`, `negate(X)`, and `imply(X,Y)`.

6. Unification in Prolog (10 Points)

For each Prolog expression, fill the variable instantiations (only for those variables used in the expression) or put a check in the "no" column if evaluation fails. The first line is already filled correctly as an example. The expressions are independent of each other. *(No partial credit, no work needs to be shown.)*

query	X =	Y =	Z =	no
`X = 3, Y is X*X.`	3	9		
`X = f(Y,Z,Z), Y = a, Z = b.`				
`X = 2, Y is X + 2, X + Z is X + Y.`				
`[[a|X], X, Y] = [Z, [b, c], [Z]].`				
`g(a, f(b,Y), [c|X]) = g(Y, f(X,a), [c, b]).`				
`[X|h(a,Y)]=[h(a)|h(a,Y)], [h(a,X)|Z]=[Y|g(g,c)].`				

7. Parameter passing (9 points)

Consider the following program in C-like syntax:

```
#include <stdio.h>

void foo (int x, int y)
{
  x = y - x;
  y = x + y;
}

int main ()
{
   int A[4], i, k=3;

   for (i = 0; i < 4; i++)
      A[i] = i + 1;

   foo (k, A[k]);
   printf("%d", k);
   for (i = 0; i < 4; i++)
      printf(", %d", A[i]);

   return 0;
}
```


What does the program output under the following different parameter passing schemes?

(a) (3 points) call-by-value:

(b) (3 points) call-by-reference:

(c) (3 points) call-by-name:

8. **Extra Credits** (10 points)

   (a) Call-by-name (6 points)
   As we studied in this class, call-by-name offers some interesting advantages over call-by-value and call-by-reference. However, call-by-name is also more challenging to implement. *Concisely* describe the *key challenge* (3 points) and *how it is tackled* (3 points).

   (b) Static vs. dynamic type checking (4 points)
   Identify *concisely* key *relative benefits* of performing static vs. dynamic type checking.